1The opinion in support of the decision is *not* binding precedent of the Board.

UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES

Ex parte JEREMY ALAN ARNOLD
and JOHN MATTHEW SANTOSUOSSO

Appeal 2007-0506
Application 09/998,511[1]
Technology Center 2100

Decided: October 30, 2007

Before JAMES D. THOMAS, LEE E. BARRETT, and JEAN R. HOMERE, *Administrative Patent Judges*.

BARRETT, *Administrative Patent Judge*.

DECISION ON APPEAL

1This is a decision on appeal under 35 U.S.C. § 134(a) from the Final Rejection of claims 1-5, 7-15, and 17-30. We have jurisdiction pursuant to 35 U.S.C. § 6(b).

We reverse.

---

[1] Application for patent filed November 30, 2001, entitled "Inheritance Breakpoints for Use in Debugging Object-Oriented Computer Programs."

BACKGROUND

The claims are directed to an apparatus, program product, and method utilizing an "inheritance breakpoint" to assist in debugging an object-oriented computer program having a method identified in a base class or interface and implemented by at least one method implementation in another class that depends therefrom. Methods may have multiple implementations which use the same name or identifier, so when a program makes a call to a particular method, it may not be readily apparent to the programmer exactly what implementation of the method will be executed as a result of the call. Creation of an inheritance breakpoint for a particular method responsive to a user request typically results in the automated generation of a breakpoint for all or a subset of the implementations of that method, freeing a user from having to manually determine what particular implementation(s) implement that method. *See* Abstract; Specification 3-4.

For example, Figure 7 illustrates a portion of a computer program 162 including a definition 164 for an interface ("MyInterface") and two definitions of classes ("MyClass1" and "MyClass2") that implement the MyInterface interface. An identification of a method "MyMethod()" is provided at line 2 of the interface definition, with implementations of the method defined in each class definition at lines 5-7 and 10-12. The classes are separate program entities (a "program entity . . . may be any of a number of different program structures supported by an object-oriented programming environment within which a method may be identified and/or defined, e.g., a class, an abstract class, an interface, etc."

Specification 8: 12-15) from the interface. The main class 170 creates a new MyClass1 or MyClass2 object (lines 17-21) depending on an input argument, and then calls the MyMethod() method on the newly created object at line 22. The user may set an inheritance breakpoint (IBP) on MyMethod() in the MyInterface interface (the "first program entity" in claim 1) and the program will halt execution when it reaches one of the two classes that implement the method ("second program entity" in claim 1). This assists the programmer in identifying what implementations of methods are being executed while debugging object-oriented computer programs. *See* Specification 18: 4-24.

Claim 1 is illustrative:

> 1. A computer-implemented method of debugging an object-oriented computer program, the method comprising:
>
> > (a) in response to user input, setting an inheritance breakpoint that is associated with a first program entity in the object-oriented computer program and a method identified in the first program entity; and
> >
> > (b) halting execution of the object-oriented computer program during debugging in response to reaching an implementation of the method defined in a second program entity in the object-oriented computer program that is different from and that depends from the first program entity.

THE REFERENCES

Coplien	US 5,093,914	Mar. 3, 1992
West	US 5,740,440	Apr. 14, 1998
Nishimura	US 5,845,125	Dec. 1, 1998

Admitted Prior Art (APA) at Specification 2 stating that it was known for breakpoints to be specified as "conditional."

THE REJECTIONS

Claims 1, 3, 4, 7-10, 13, 15, 17, 18, 20, 22-24, and 26-30 stand rejected under 35 U.S.C. § 102(b) as being anticipated by Nishimura.

Claims 2, 5, 14, 19, and 21 stand rejected under 35 U.S.C. § 103(a) as unpatentable over Nishimura and Coplien.

Claim 11 stands rejected under 35 U.S.C. § 103(a) as unpatentable over Nishimura and West.

Claims 12 and 25 stand rejected under 35 U.S.C. § 103(a) as unpatentable over Nishimura and the APA.

DISCUSSION

Content of Nishimura

Nishimura discloses a debugger for an object-oriented computer program. A conventional debugging system cannot debug an object-oriented program efficiently because it processes the program based on the location within the program, not based on objects (col. 2, ll. 54-57) and has the problems described at column 2, line 60 to column 4, line 40.

Nishimura allows a user to set a breakpoint on an object basis or at a desired location in an object (col. 14, ll. 55-65).

"An 'object-type breakpoint' is used to break the execution of a user-specified object when an operation is performed on the object . . . ." (Col. 14, ll. 65-67). When the user sets an object-type breakpoint, the breakpoint setting section 16 sets a breakpoint in all of the public functions of the class and of the indirect base classes (col. 15, ll. 12-17), where "all [of] the derived (higher level) classes, including the base class of a class, the base class of the base class, and so forth, are called indirect base classes" (col. 2, ll. 33-37). "By setting breakpoints in all those functions, the user can determine whether a break occurs when one of those functions is called." (Col. 15, ll. 17-19.)

"[A]n 'address-type breakpoint' is used to break the execution of an object when control reaches a user-specified location in a user-specified object (that is, at a location in a particular member function or in a particular location in an inherited member function)." (Col. 15, ll. 1-5.) By setting the breakpoint only in a function or at an address within the object whose breakpoint is to be checked, the breakpoint does not break execution unless control reaches the corresponding address, increasing debugging efficiency (col. 15, ll. 6-11).

Anticipation

Claim 1 requires "halting execution of the object-oriented computer program during debugging in response to reaching an ***implementation*** of the

method ***defined*** in a second program entity in the object-oriented computer program that is different from and that ***depends from*** the first program entity" (emphasis added). Appellants argue that this limitation requires two things: (1) execution is halted in response to hitting an implementation of a method in a second program entity that "depends from" the first program entity (App. Br. 6; Reply Br. 2-3); and (2) the "implementation" must be "defined" within the second program entity itself, rather than being inherited from a parent program entity (most clearly argued at Reply Br. 2-4).

The main issue argued is whether Nishimura discloses that the "implementation" of the method is "defined" in a second program entity which "depends from" a first program entity. The Examiner's position is:

> "[A]n implementation of the method defined in a second program entity" as claimed, reads on an inherited function (i.e., a function **defined** in the second program entity **as inherited** from the first program entity) of Nishimura since an inherited function is inherently **defined** in the second program entity to be the same as that **defined/identified** in the base class (i.e., "first program entity").

(Ans. 10.)

Appellants argue that "[t]he arguments presented by the Examiner in the Examiner's Answer are fundamentally premised on the incorrect assumption that Applicants' claims read on the situation where a breakpoint is set on a method defined in a parent class and execution is halted when that method is executed by an object instance of a subclass that has inherited the method, but where the actual implementation of the method that is hit is defined in the parent class, but not in the subclass" (Reply Br. 3-4).

We respectfully disagree with the Examiner's interpretation of Nishimura as meeting the limitation that the "implementation" of the method is "defined" in a second program entity which "depends from" a first program entity. Where the implementation of a function is defined in the base class, that is where the implementation is defined, not in the subclass, which merely inherits the function and its definition. The program instructions implementing the function are in the base class, not in the subclass. The Examiner's interpretation that the inherited function is inherently defined to be the same as that defined in the base class, while creative, is not considered to be a fair and accurate statement.

Nishimura does not disclose that the "implementation" of the method is "defined" in a second program entity which "depends from" a first program entity. The implementations of methods in a subclass (second program entity) are in the base class (first program entity) upon which the subclass directly or indirectly depends. Accordingly, the anticipation rejection of claim 1 and its dependent claims 3, 4, and 7-10 is reversed.

Independent claims 13, 15, 18, 26, 28, and 30 have limitations corresponding to those discussed in claim 1 and are not anticipated by Nishimura for the reasons stated with respect to claim 1. The anticipation rejection of claims 13, 15, 18, 26, 28, and 30, and their dependent claims 17, 20, 22-24, 27, and 29, is reversed.

Obviousness

The Examiner finds that Nishimura does not disclose that the first program entity is an "interface . . . in which is identified a method," and wherein the second program entity is a "class . . . that implements the interface," as recited in claim 14 (Final Rejection 7). The Examiner finds that Coplien discloses debugging an object-oriented computer program having a first program entity comprising an abstract class and a second program entity that identifies the method, referring to Figure 5 and associated text (*id.*). The Examiner further finds that Coplien discloses setting a breakpoint in a function of the first program entity and halting execution during debugging in response to reaching an implementation of the method defined in the second program entity, referring to column 8, lines 45-53 (*id.*). The Examiner concludes that it would have been obvious to replace the first program entity in Nishimura with an "abstract class which would produce the expected result with reasonable success" (*id.*).

Appellants argue that Coplien merely discloses the general concept of an interface-implementation relationship in object-oriented programming, but falls short of disclosing halting execution of a program upon reaching an implementation of a method identified in an interface with which a breakpoint is associated (App. Br. 15).

Coplien discloses that object-oriented programming focuses on applications made up of objects, instances of abstract data types. An abstract data type (ADT) or class contains a template describing what data fields will be in a variable of that type and also contains (semantically) the

definitions of functions tightly associated with that type, where the functions implement the operations on that type and are called methods or member functions (col. 6, ll. 19-35). Objects are created from a class by a process called instantiation (col. 6, ll. 36-43). All objects of a given type (ADT) share their functions (col. 8, ll. 16-17). Figure 5 of Coplien discloses an ADT for the class Window having data for position and size and functions for move, refresh, create, and delete. The functions for refresh, create, and delete are not implemented in the Windows class. A class XWindow is used to implement the semantics for manipulating a window created using the X window system (col. 8, ll. 62-64). It contains declarations of data that hold information pertinent to the details of maintaining windows (pixel depth, color, etc.) and defines the functions to do what needs to be done to X windows (refresh, create, delete, etc.) (col. 8, ll. 64-68).

There is no question, and Appellants admit, that Coplien discloses the general concept of an interface-implementation relationship in object-oriented programming. Appellants do not allege that they invented the interface-implementation programming structure illustrated in their Figure 7. However, we do not agree with the Examiner's finding that Coplien discloses setting a breakpoint in a function of the first program entity and halting execution during debugging in response to reaching an implementation of the method defined in the second program entity. (If it did, it would in fact anticipate.) Column 8, lines 45-53, which the Examiner relies on, only states that when there are two different objects of the same class, modifications to their shared text to insert a breakpoint will cause

what is done in the name of one object to interfere with the others by causing the specified breakpoint to happen for all of them. This discloses breakpointing a method and halting execution when the method is executed in the object, but does not disclose breakpointing a method in the base class and halting execution of a program upon reaching an implementation of a method identified in an interface with which a breakpoint is associated. Thus, neither Nishimura nor Coplien discloses halting execution of a program upon reaching an implementation of a method identified in an interface with which a breakpoint is associated. The rejection of claim 14 is reversed.

Coplien likewise fails to cure the deficiencies in independent claims 1 and 18. Therefore, the rejection of claims 2, 5, 19, and 21 is reversed.

West and the APA have not been applied for limitations that would cure the deficiencies of independent claims 1 and 18. Therefore, the rejections of claims 11, 12, and 25 are reversed.

CONCLUSION

The rejections of claims 1-5, 7-15, and 17-30 are reversed.

REVERSED

MAT

WOOD, HERRON & EVANS, L.L.P. (IBM)
2700 CAREW TOWER
441 VINE STREET

CINCINNATI, OH 45202